ISSN 1009 - 3095 *Journal of Zhejiang University(SCIENCE)* V.1, No.4, P.384 - 387, Oct. - Dec., 2000
http://www.chinainfo.gov.cn/periodical; http://www.zju.edu.cn;
http://lib.zju.edu.cn/English; http://www.zjupress.com

RESEARCH ON COMPUTER-AIDED PROTOTYPING SYSTEM AND SOFTWARE EVOLUTION*

YING Jing (应 晶)

(*Department of Computer Science, Zhejiang Univerisity, Hangzhou 310027, China*)

Received Jan.12, 1999; revision accepted May.26, 2000

Abstract: This paper presents a survey on the status of current research on software evolution and computer-aided prototyping system; and also puts forward extension on computer-aided software evolution and prototyping, to provide a framework for integrating software evolution activities.

Key words: software evolution, prototyping
Document code: A **CLC number:** TP311.52

SOFTWARE EVOLUTION

Software evolution has been addressed from many viewpoints. These include management issues such as which parts of a system should be rebuilt, configuration issues such as keeping track of many versions of the same system, testing issues such as determining which test cases could be influenced by a given change, requirements issues such as determining when the assumptions underlying a system specification are no longer valid, code restructuring issues, performance improvement issues, and much more. Many of these problems do not have accepted or validated formal models, and the application of formal methods is less obvious than in more mature areas, such as proving program properties. Software evolution includes all the activities that change a software system, as well as the relationships among those activities (Luqi, 1997). Evolution encompasses activities ranging from adjusting requirements to updating working systems, including responses to requirements changes, improvements to performance and clarity, bug repair, version and configuration control, documentation, testing, coding generation, and the overall organization of the development process.

Two related problems here are, traceability, which is the problem of tracing design decisions, or fragments of specification or code text, back to the requirements that they are supposed to meet, and the difficulty of maintaining the dependencies among components in a large software system development effort. Current capabilities for software evolution are inadequate. The most visible symptoms of the problem are the large backlog of requested changes, long delays, high error rates, and an alarming incidence of failure to complete changes. Some causes of these problems are missing information, intellectual overload, and primitive tool support.

The capabilities of current tools for supporting software evolution are limited by lack of formal models and theoretical bases. Formal models can lead to tools that enhance the capabilities of people to reliably change complex software systems. Such tools will make it easier to formalize and record more of the dependencies in a design, and to provide more sophisticated kinds of decision support based on this information.

COMPUTER-AIDED PROTOTYPING

In the software project development and evolution cycles, prototyping techniques will allow for the visualization of project requirements and achieve their concurrence among interested participants, and with that visualization comes the understanding of the requirements necessary to formulate the design. Prototyping can also be used to explore design alternatives using the derived requirements.

Prototyping is a process of quickly building and evaluating a series of prototypes, where a

* Project supported by NSFC(69703005) and Zhejiang Provincial Natural Science Foundation(697006).

prototype is a concrete executable model of selected aspects of a proposed system. In prototyping, evolution activities are interleaved with development, and continue after the delivery of the initial version. Four advantages are embodied in the prototyping process:

1) Prototyping allows developing and presenting requirement model.

2) Prototyping provides the visualization and the validation of system requirements.

3) Prototyping provides a method for accessing the feasibility of alternative design approaches.

4) Prototyping allows for rapid evaluation of the performance characteristics of designs.

Prototyping can be used in a software evolution environment in the same manner as in a development environment and the benefits of costs reduction are similarly realized. The traditional software evolution activity, which addresses the maintenance of the program after the development is complete, has the responsibility of developing the design for unmet, new or changed software requirements, is responsible for the system meeting current or new performance requirements, and also must evaluate the current software for design deficiencies. When system design changes to a fielded system are proposed, the changes that need to be incorporated to the existing system to evaluate each of several design alternatives may be extensive and therefore costly. Prototyping techniques can expand the maintenance programmers capability by providing the best alternative to the choice of design alternatives, can allow for exploration of possible design alternatives and will provide increased understanding of the new requirements and confidence in the selected design approach. Prototyping can be used to approximate new interface and simulate the interaction between program elements.

CAPS SYSTEM

1. Overview of CAPS

The Computer-aided Prototyping System (CAPS) is a software engineering collection of tools for developing prototypes of real-time systems (Luqi, 1988a). It is useful for requirement analysis, feasibility studies, and the design of large embedded systems. CAPS is based on the prototype system description language (PSDL) (Luqi, 1988b), which provides facilities for modeling timing and control constraints within a software system. CAPS is a development environment, implemented in the form of an integrated collection of tools, linked together by a user interface.

CAPS consists of an integrated set of tools that help design, translate and execute prototypes. These include a graph data model for evolution, evolution control system, change merging facility, automatic generators for schedule and control code, and automated retrievals for reusable components (Luqi, 1990).

2. Evolution control system

CAPS comprises an evolution control system (ECS) that provides automated assistance for the software evolution process in an uncertain environment where designer tasks and their properties are always changing. We view an ECS as the agent that keeps track of proposed, ongoing, and completed changes to a software system. It provides automated assistance to the software evolution manager to help make the right decisions. It automatically propagates change consequences by constructing the set of possibly affected modules.

An ECS has two main functions. The first is to control and manage evolving software system components and the second is to control and coordinate evolution team interactions in a way that maximizes the concurrent assignment and meets management constraints such as deadlines and precedence (planning and scheduling software evol-ution tasks which we refer to as evolution steps).

3. Graph model

The graph model is represented as a directed acyclic graph $G = [C, S, I, O]$, where

C: software component nodes;

S: evolution step nodes;

$I \subseteq C \times S$: input relation between the system components and the evolution steps;

$O \subseteq C \times S$: output relation between the evolution steps and the components.

C and S are the two kinds of nodes. Each node has a unique identifier. C and S nodes alternate in each path that has only I and O edges.

The CAPS graph model is a data graph

model for evolution that records dependencies and supports automatic project planning, scheduling, and configuration management. According to this model, the evolution process of a software system is represented by a graph that at any given moment models the current and the past state of the software system. A typical instance of that graph consists of software objects that comprise the system configuration and the evolution activities (steps) applied to these objects. The graph model views a software evolution process as a partially ordered set of steps. Each change in the system design from the moment it is proposed is performed within the context of one or more steps. Steps have states that reflect the dynamic progression of the change from the moment it is proposed until it is completed or abandoned. When rejected, history of the activity remains in the project database. When completed, a step outputs new version or versions of the subject software component that underlies the change.

EXTENSION TO CAPS

This extension provides a framework for integrating software evolution activities. It aims at supporting the computer-aided software evolution based on graph model to enable CAPS support practical software development through managing software evolution in a rapidly evolving system. We make certain extensions to CAPS:

1) extend the functionality and scope of the evolution control system to include the requirements evolution and analysis process within the context of a formal model. With our extensions the early part of the prototyping process becomes more formal.

2) extend the graph model to enable linking requirements to the system design and implementation to provide the automated support to expose consequences of the requirements changes on the system design. The extensions and enhancements we develop support recording, analyzing, and resolving customers' concerns.

3) Inference rules enable CAPS to provide the automated support for extracting the dependencies among components, specifying and enforcing constraints, generating and evaluating alternative requirement changes.

The main objective is to design an evolution control system that can provide automated assistance for the computer-aided software evolution process based on CAPS and graph model. The main techniques we would address are requirements gathering, project inter-dependencies, changing technology, project scheduling and resource management. We view an evolution control system as the agent that keeps track of proposed, ongoing, and completed changes to a software system. It provides automated assistance to the software evolution manager to help make the right decisions. It automatically propagates change consequences by defining the set of possibly affected modules.

The evolution graph model is enhanced and extended into a conceptual model that incorporates the modeling of the requirements analysis and evolution process. This enables the extension of the project control functionality in CAPS to include requirements evolution and the automated link of this evolution to the design and implementation levels of the system prototype.

This model has provisions for representing the customers' responses to the demonstrated systems, synthesizing issues from these responses, analyzing the affected requirements, and assisting in identifying alternatives available to resolve open issues. This is done within the context of analysis and design activities generated automatically. These activities assist managers in controlling and coordinating the project progress and implementing plans.

Our emphasis is laid on the dependencies among components. The traceability problem is the difficulty of maintaining the huge mass of dependencies among the many objects produced by a large software system development effort. Often these objects are not adequately defined; for example, module boundaries may be incorrectly drawn or not even explicitly stated at all and interfaces may be poorly drawn or badly documented. Without using representations for the objects involved, formal models for the dependencies, and tool support for managing them, it is impossible to know what effect a change will have, and in particular, to know what other objects may have to be changed to maintain consistency. Particular challenges include formalizing dependencies and developing methods for calculating dependencies and propagating the implications of changes. This approach is intended to support,

by linking related objects, both the context of requirements decisions and their traceability.

In support of teamwork, the system identifies tasks that can be executed concurrently by members of the design team. When a designer whose expertise field and level match the task requirements available, the step is assigned to him and an automatic transition from scheduled to assigned task occurs. Identification of concurrent tasks is computed partially from the dependency graph by considering the relationships between the graph parts. For example, a rule can be represented as:

If all $(s_1, s_2 : S :: ((s_1, s_2) \in E)$
Then (s_1 precedes s_2)

We propose a new mechanism based on the inference rules to support inferring additional dependencies from the given, specifying and enforcing constraints, and establishing relationships. The system infrastructure provides the support to the individuals involved in the software design process, including stakeholders, software project managers, software analysts, and software designers.

DISCUSSION AND CONCLUSION

In this paper, we give a survey on software evolution and computer-aided prototyping. According to the software experiences, software evolution will play an important role in the future software development, and computer-aided prototyping will become a practical technique in the evolution process. CAPS consists of an integrated set of tools that help design, translate and execute prototypes. It includes a graph data model for evolution, evolution control system, change merging facility, and automated retrievals for reusable components. It supports the prototyping modeling process quite well. On these bases, we put forward the extension on CAPS system.

The extension is to develop and integrate a requirement dependencies mechanism which will propagate changes not only between the specification and implementation components, but also include requirements documents on-line where any change to a software system's requirements automatically triggers proposed changes to corresponding specific-ation modules and subsequently, the implementation modules. We have focused on software evolution because large-scale software development is dominated by constantly changing requirements and the difficulties of implementing the ensuing changes. Our experience indicates that formal models can provide useful decision support for software evolution. The final goal is to develop better computer aid for software evolution.

ACKNOWLEDGEMENTS

The authors would like to thank CAPS group members for their helpful discussion and comments on this research.

References

Luqi, Ketabchi, M., 1988a. A Computer-aided Prototyping System. *IEEE Software*, **5**(3):18 - 28.

Luqi, Berzins, V., 1988b. A Prototyping Language for Real-Time Software. *IEEE Trans. Software Eng.*, **14** (10):1035 - 1051.

Luqi, 1990. A Graph Model for Software Evolution. *IEEE Trans. Software Eng.*, **16**(8):688 - 696.

Luqi, Goguen, J.A., 1997. Formal Methods: Promises and Problems, *IEEE Software*, **14**(1):28 - 35.